Name: Jai Darshan S

Dept: AI&DS

Date: 11/11/2024

Practice set – 2

1. 0-1 knapsack problem:

Java code:

public class Knapsack {

    public static int result(int capacity,int[] wt,int[] val,int n){

        int[][] dp=new int[n+1][capacity+1];

        for (int i=1;i<=n;i++){

            for (int w=1;w<=capacity;w++){

                if (wt[i-1]<=w){

                    dp[i][w]=Math.max(dp[i-1][w], dp[i-1][w-wt[i-1]]+val[i-1]);

                }else{

                    dp[i][w]=dp[i-1][w];

                }

            }

        }

        return dp[n][capacity];

    }

    public static void main(String[] args) {

        int[] val1={1,2,3}, wt1={4,5,1};

        int[] val2={1,2,3}, wt2={4,5,6};

        int[] val3={10,40,30,50}, wt3={5,4,6,3};

        int c1=4,c2=3,c3=5;

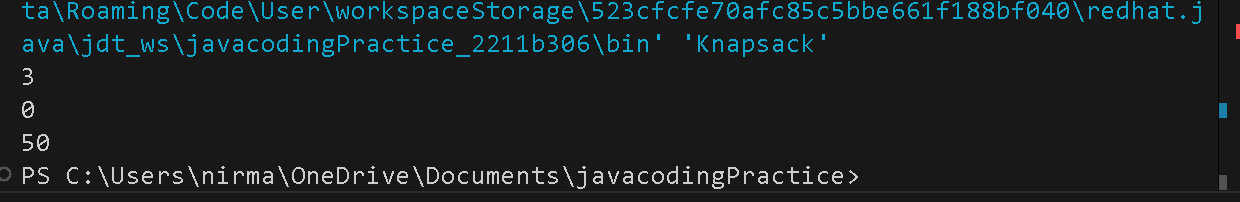
        System.out.println(result(c1, wt1, val1, val1.length));

        System.out.println(result(c2, wt2, val2, val2.length));

        System.out.println(result(c3, wt3, val3, val3.length));

    }}

Output:



Time complexity: O(nxCapacity)

Space complexity: O(nxCapacity)

2. Floor in sorted array:

Java code:

public class FloorinSortedArray {

    public static int floor(int[] arr,int k){

        for (int i=arr.length-1;i>=0;i--){

            if (arr[i]<=k){

                return i;

            }

        }

        return -1;

    }

    public static void main(String[] args) {

        int[] arr1={1, 2, 8, 10, 11, 12, 19};

        int[] arr2={1, 2, 8};

        int k1=0,k2=5,k3=1;

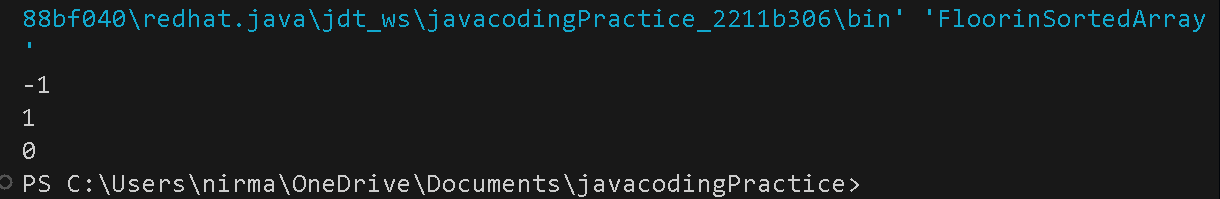
        System.out.println(floor(arr1, k1));

        System.out.println(floor(arr1, k2));

        System.out.println(floor(arr2, k3));

    }

}

Output:

Time complexity: O(n)

Space complexity: O(1)

3. Check equal arrays:

Java code:

import java.util.HashMap;

import java.util.Arrays;

public class CheckEqualArray {

    public static boolean check(int[] arr1,int[] arr2){

        if (arr1.length!=arr2.length){

            return false;

        }

        HashMap<Integer,Integer> map=new HashMap<>();

        for (int i:arr1){

            map.put(i, map.getOrDefault(i,0)+1);

        }

        for (int j:arr2){

            if (!map.containsKey(j)){

                return false;

            }

            map.put(j, map.get(j)-1);

            if (map.get(j)<0){

                return false;

            }

        }

        for (int k:map.values()){

            if (k!=0){

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        int[] arr1={1,2,5,4,0};

        int[] arr2={2,4,5,0,1};

        int[] arr3={1,2,5};

        int[] arr4={2,4,15};

        System.out.println(Arrays.toString(arr1)+", "+Arrays.toString(arr2)+": "+check(arr1, arr2));

        System.out.println(Arrays.toString(arr3)+", "+Arrays.toString(arr4)+": "+check(arr3, arr4));

    }

}

A screen shot of a computer code

Description automatically generatedOutput:

Time complexity: O(n)

Space complexity: O(n)

4. Palindrome linked list

Java Code:

import java.util.Stack;

class Node{

    int data;

    Node next;

    Node(int d){

        data=d;

        next=null;

    }

}

public class PalindromicLinkedList {

    public static boolean palindrome(Node head){

        Node curr=head;

        Stack<Integer> st=new Stack<>();

        while (curr!=null){

            st.push(curr.data);

            curr=curr.next;

        }

        while (head!=null){

            int a=st.pop();

            if (head.data!=a){

                return false;

            }

            head=head.next;

        }

        return true;

    }

    public static void main(String[] args) {

        Node head=new Node(1);

        head.next=new Node(2);

        head.next.next=new Node(2);

        head.next.next.next=new Node(1);

        Node head2=new Node(1);

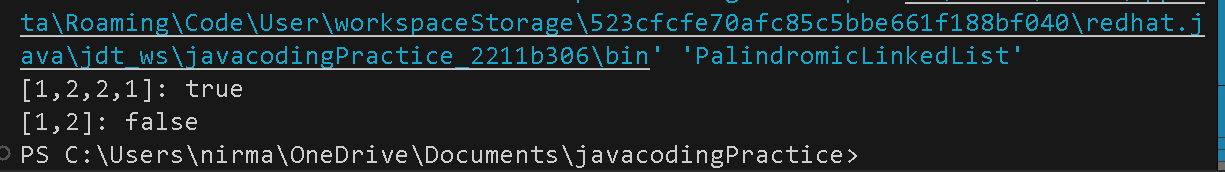
        head2.next=new Node(2);

        System.out.println(palindrome(head));

        System.out.println(palindrome(head2));

    }

}

Output:

Time complexity: O(n)

Space complexity: O(n)

5. Balanced tree check:

Java code:

class Tree{

    int data;

    Tree left,right;

    Tree (int d){

        data=d;

        left=null;

        right=null;

    }

}

public class BalancedTreeCheck {

    public static int height(Tree root){

        if (root==null){

            return 0;

        }

        int lefth=height(root.left);

        if (lefth==-1){

            return -1;

        }

        int righth=height(root.right);

        if (righth==-1){

            return -1;

        }

        if (Math.abs(lefth-righth)>1){

            return -1;

        }

        return Math.max(lefth, righth)+1;

    }

    public static boolean checkbalance(Tree root){

        if (height(root)!=-1){

            return true;

        }else{

            return false;

        }

    }

    public static void main(String[] args) {

        Tree t1=new Tree(1);

        t1.left=new Tree(2);

        t1.left.right=new Tree(3);

        System.out.println(checkbalance(t1));

        Tree t2=new Tree(10);

        t2.left=new Tree(20);

        t2.right=new Tree(30);

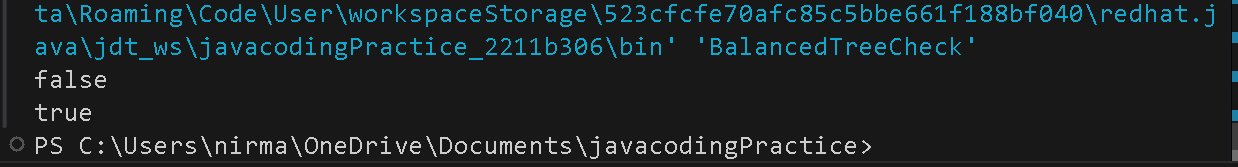
        t2.left.left=new Tree(40);

        t2.left.right=new Tree(60);

        System.out.println(checkbalance(t2));

    }

}

Output:

Time complexity: O(n)

Space complexity: O(n)

6. Triplet sum in array

Java code:

import java.util.Arrays;

public class TripleSumOfArray {

    public static boolean sum(int[] arr, int x){

        Arrays.sort(arr);

        for (int i=0;i<arr.length;i++){

            int left=i+1;

            int right=arr.length-1;

            while (left<right){

                int s=arr[i]+arr[left]+arr[right];

                if (s==x){

                    return true;

                }else if (s<x){

                    left++;

                }else{

                    right--;

                }

            }

        }

        return false;

    }

    public static void main(String[] args) {

        int[] arr1={1,4,45,6,10,8};

        int[] arr2={1,2,4,3,6,7};

        int[] arr3={40,20,10,3,6,7};

        int k1=13,k2=10,k3=24;

        System.out.println(sum(arr1, k1));

        System.out.println(sum(arr2, k2));

        System.out.println(sum(arr3, k3));

    }

}

A screen shot of a computer

Description automatically generatedOutput:

Time complexity: O(n^2)

Space complexity: O(1)